Annotation by Transformation for the Automatic Generation of Content Customization Metadata

Masahiro Hori, Kouichi Ono, Teruo Koyanagi, and Mari Abe

IBM Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi
Kanagawa-ken, 242-8502, Japan
{horim, onono, teruok, maria}@jp.ibm.com

Abstract. Users are increasingly accessing the Internet from mobile devices as well as conventional desktop computers. However, it is not reasonable to expect content authors to create different data presentations for each device type, but the content source should be reused across multiple delivery contexts whenever possible. The objective of this research is to develop a supporting tool for the presentation customization that follows after the content specialization in device-independent authoring. This paper presents a tool that automatically generates content customization metadata on the basis of users' editing operations toward the desired results of the customization. A prototype of the metadata generator was developed for the generation of page-clipping annotations to be used for an annotation-based transcoding system.

1 Introduction

As more and more Web-enabled personal devices are becoming available for connecting to the Internet, the same Web content needs to be rendered differently on different client devices, taking account of their physical and performance constraints such as screen size, memory size, and connection bandwidth. For example, a large full-color image may be reduced with regard to size and color depth, removing unimportant portions of the content. Such device adaptation is exploited for Web documents delivered over HTTP, and results in better presentation and faster delivery to the client device. Device adaptation is thus crucial for transparent Web access under different delivery context, which may depend on client capabilities, network connectivity, or user preferences [2,3,9,22]. It must be noted that the Web documents refer not only to existing HTML pages, but also to XML/XHTML documents that may be generated from a content source.

It is not reasonable to expect content authors to create multiple versions of the same Web content, each specialized for a different delivery context. Such multiple authoring encounters the *M times N problem*, namely, an application composed on M pages to be accessed via N devices requires M x N authoring steps, and results in M x N presentation pages that must be maintained [18]. The

F. Mattern and M. Naghshineh (Eds.): Pervasive 2002, LNCS 2414, pp. 267–281, 2002.

key challenge here is to enable Web content to be delivered through a variety of access mechanisms with minimum effort.

The objective of this research is to develop technologies to be used for presentation customization in a device-independent authoring environment. The customization requires additional data about the ways of modifying the presentation, and it is assumed in this study that metadata or annotations[1] are exploited by a runtime engine for the content customization at content delivery time. An annotation is a remark attached to a document, and declares properties that qualify a particular portion of a target document. In addition, annotations may indicate structural changes for the annotated portion of a target document. In order to clarify the distinction of these two roles, we call the former *assertional annotations* and the latter *transformational annotations*. It is important to note that this distinction is not exclusive, because every annotation is intrinsically an assertion.

It is straightforward for annotation authors to indicate a location to be annotated and create an assertion as annotation content. This is an approach that we call *annotation by assertion*, and is adopted by existing annotation editors [1,5, 8,12,14,20,23]. On the other hand, for transformational annotations, it is easier for the authors to modify a target document toward the desired results of the customization, rather than to indicate the ways of modifications declaratively as assertional annotations. This is a basic idea behind an approach what we call *annotation by transformation*, which was originally proposed for the automatic generation of document transformation rules [15]. In this paper, we present a tool that follows the annotation by transformation approach, and automatically generates content customization metadata on the basis of users' editing operations for the content customization.

In the next section, we present a perspective on device independent authoring, and briefly explains the usefulness of the annotation by transformation approach for supporting content customization. Section 3 presents an annotation-based content adaptation system, and introduces a content customization language for annotation-based document clipping. Section 4 explains approaches to metadata authoring taking account of the distinction between assertional and transformational annotation. Finally, we present our prototype of the metadata generation tool, and emphasize the complementary roles between the annotation by transformation and annotation by assertion approaches.

2 A Perspective on Device Independent Authoring

The ultimate goal of device-independent authoring is to realize a *single authoring* environment [18], in which presentation suitable for each delivery context is generated from an abstract interaction model that does not depend on any particular delivery context. The single authoring, which is illustrated in Fig. 1, consists of not only the content *specialization* process, but also a content *customization* process.

[1] Metadata and annotations are used interchangeably in this study.

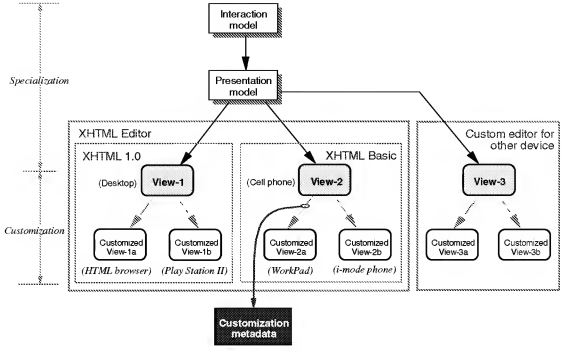

Fig. 1. Illustration of content customization in the single-authoring process

In the authoring for the specialization, content authors need to think abstractly by going back and forth between the device-neutral interaction model and a presentation model specialized for a class of document representation. The specialization thus entails document type conversion from the schema for an interaction model toward another schema for a presentation model. The customization, on the other hand, is characterized as document modification under the same document type or schema.

The content customization allows content authors to further elaborate the specialized presentation to be more specific to the delivery context, in order to meet criteria such as ease of use and quality of presentation [6]. Authoring for the customization is largely concerned with details of the presentation, which depend on individual device capabilities and users' preferences. In such a situation, it is desirable for content authors to create an expected result interactively with an example, rather than working abstractly without any concrete example.

Learning an abstract language and writing programs are not easy tasks for most people. However, if a person knows how to perform a task to be executed by a computer, perhaps the person's knowledge can somehow be exploited for the creation of a program to perform the task. This is the motivation behind *programming by example* [17], which is also called *programming by demonstration* [4]. Programming by example would be a natural approach to creating the presentation customization metadata for page designers or novice programmers, because users need only work with examples of how to transform a page at hand, and are given automatically generated metadata that can replicates the same transformation. Fig. 2 illustrates the idea of generating content customization

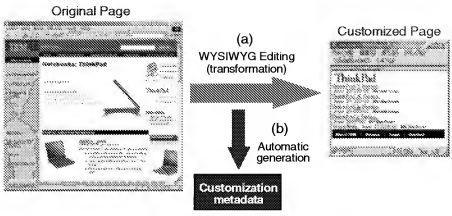

Fig. 2. Illustration of annotation generation by example

metadata (e.g., page-clipping annotation) as results of WYSIWYG editing of an original page.

On the basis of the idea of the metadata generation from users' operation histories, we have already developed a prototype of a metadata generation tool XSLbyDemo [21,15] as an add-on module for a commercially available WYSIWYG HTML editor. The XSLbyDemo generates document transformation rules for the XSL Transformation Language (XSLT) [28]. However, the XSLbyDemo could only function within the HTML editor, even though the generation functions were designed to be independent of any specific editor programs.

The metadata generator presented in this paper was developed as a plug-in module for an open tool integration platform [25], so that arbitrary editors can be chosen for the content customization authoring. In addition, the metadata generator is designed as a tool framework to be customized for a class of metadata languages. In the next section, annotation-based page clipping is introduced and an example of the page-clipping annotation is explained. The metadata generator specialized for the page-clipping annotation language is then presented in the section 4.

3 Annotation-Based Content Adaptation

Web-content metadata or annotations have a variety of applications [16], which can be categorized into three types: discovery, qualification, and adaptation of Web contents. The primary focus of this research is on the Web-content adaptation, and the role of annotation is to characterize ways of content adaptation rather than to describe individual contents themselves.

3.1 Annotation Framework

Annotations can be embedded into a Web document as an inline annotation. Inline annotations are often created as comments or extra attributes of document elements. Because of its simplicity, inline annotation has been often adopted as a

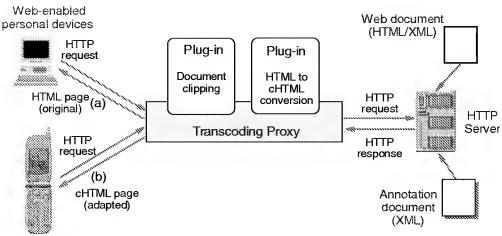

Fig. 3. Overview of an annotation-based transcoding

way of associating annotation with HTML documents [19,22,10]. An advantage of the inline annotation approach is the ease of annotation maintenance without the bookkeeping task of associating annotations with their target document. The inline approach, however, requires annotators to have document ownership, because annotated documents need to be modified whenever inline annotations are created or revised. Furthermore, mixing of contents and metadata is not desirable with regard to the design consideration that separates content from presentation.

The external annotation approach [11,14,20,24], on the other hand, does not suffer from the issues related to the document ownership. Moreover, the most important point of the external annotation is that this approach facilitates the sharing and reuse of annotations across Web documents. Since an external annotation points to a portion of a Web document, the annotation can be shared by Web documents that have the same document fragment.

Annotations provide additional information about Web contents, so that an adaptation engine can make better decisions on the content re-purposing. The role of annotations is to provide explicit semantics that can be understood by a content adaptation engine [13]. Fig. 3 depicts an overview of an annotation-based transcoding process. Upon receipt of a request from a client, a Web document is retrieved from a content server. Taking account of the capabilities of the client specified in the HTTP request header, a transcoding proxy selects one or more transcoding modules, which are indicated as plug-ins in Fig. 3. When a selected transcoding module requires an annotation document, an annotation file is also retrieved from a content server, which may or may not be the same server that retrieved the Web document. The transcoding module may simply return the original document, if a client agent has the rendering capabilities compatible with ordinary desktop computers [Fig. 3 (a)]. Alternatively, the original document may be returned with modification, so that the original content can fit into a small screen device [Fig. 3 (b)]. The decisions about the content adaptation are made taking account of the client capabilities specified in the HTTP request header.

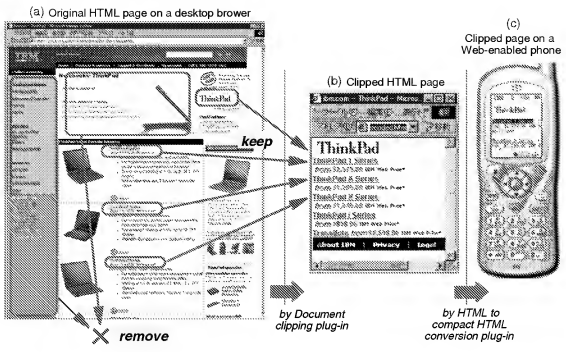

Fig. 4. Illustration of content-adaptation process

3.2 Annotation-Based HTML-Page Clipping

Fig. 4 shows a catalog page of notebook computers. The catalog page contains a lot of information, such as details of the product specification, a search field, and numerous links to other areas of the site that might be of interest to the user. However, it may be necessary to deliver portions of this page for users to access through a Web-enabled phone rather than a desktop browser. In such a case, the images and nested HTML tables prepared for a nicely laid out page are a hindrance rather than help. The sheer amount of information becomes unwieldy in the small display, and potentially expensive depending on the user's wireless service.

Content adaptation as illustrated in Fig. 4 can be done, for example, by using an annotation-based page-clipping engine [24]. At content delivery time, the page-clipping engine may modify the original document with reference to page-clipping annotations and client profiles sent over HTTP. The main idea in the page-clipping annotation language is the notion of a clipping state. By using `<keep>` and `<remove>` elements in the annotation descriptions, users can specify the clipping state to indicate whether the content being processed should be preserved or removed.

As a simple example, an HTML page and its clipped results are shown in Fig. 5. In this example, the header and the first paragraph are preserved as shown in Fig. 5(a). The table element is modified by deleting the third column and the second row. The cell-padding attribute of the table is increased, so that each table cell can be provided with margin space [Fig. 5(b)]. In addition, the whole of the second paragraph is removed as shown in Fig. 5(c).

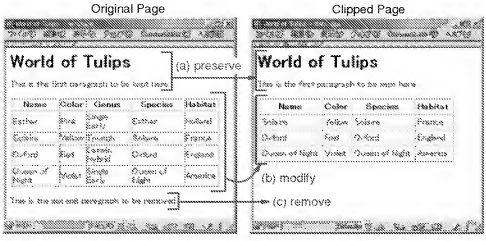

Fig. 5. Simple example of an HTML page and its clipped result

Fig. 6 shows an example of annotation document that allows the page clipping illustrated in Fig. 5. The `<description>` element prescribes a unit of an annotation statement in the annotation language. The `target` attribute is set to an XPath expression [27], and identifies the node on which the annotation will be applied, and the `take-effect` attribute indicates whether the annotation is applied before or after the target node. By specifying `target="/HTML[1]/BODY[1]/*[1]"` as in Fig. 6(a), the clipping state is activated after the first element after the first `<BODY>` element, which in this case is an `<H1>`. The `<keep>` element in Fig. 6(a) indicates that all the document elements encountered are preserved, until otherwise instructed by another annotation statement.

The clipping state is changed to 'remove' just before the second `<P>` element [Fig. 6(c)], and changed back to 'keep' after the `<P>` element [Fig. 6(d)]. As results, the second paragraph element indicated by `"/HTML[1]/BODY[1]/P[2]"` is removed while preserving the elements just before and after the removed element.

Since HTML tables can often be complex elements to clip, the annotation language provides special-purpose elements to make table clipping easier. The `<row>` and `<column>` elements allow user to clip rows and columns without relying on complicated XPath expressions. The table-clipping elements are used in the description shown in Fig. 6(b). This description sets the clipping state to 'keep' just before the first table element, and also changes the value of `cellpadding` attribute to 4 by using the `<insertattribute>` element. The `name` attribute of `<insertattribute>` can be specified with an arbitrary name of an attribute available for a target document.

In addition, the description element [Fig. 6(b)] declares that the third column, which is indicated by the `index` value of the `<column>` element, is discarded, while the remaining columns are preserved. Note here that the wildcard character to indicate multiple columns (`index="*"`). If a wildcard is specified, all rows (or columns) will be affected, except for those specifically indicated by a separate `<row>` (or `<column>`) element. So, all rows but the second will be preserved for the target table.

```
<?xml version='1.0' ?>
<annot version="2.0">
  <!-- (a) Set the default clipping state to 'keep' -->
  <description take-effect="before" target="/HTML[1]/BODY[1]/*[1]">
    <keep/>
  </description>

  <!-- (b) Remove a column and a row of the first table, -->
  <!--     and change a cellpadding attribute value      -->
  <description take-effect="before" target="/HTML[1]/BODY[1]/TABLE[1]">
    <keep/>
    <table>
      <column index="3" clipping="remove"/>
      <column index="*" clipping="keep"/>
      <row index="2" clipping="remove"/>
      <row index="*" clipping="keep"/>
    </table>
    <insertattribute name="cellpadding" value="4"/>
  </description>

  <!-- (c) Set the clipping state to 'remove' -->
  <description take-effect="before" target="/HTML[1]/BODY[1]/P[2]">
    <remove/>
  </description>
  <!-- (d) Set the clipping state back to 'keep' -->
  <description take-effect="after" target="/HTML[1]/BODY[1]/P[2]">
    <keep/>
  </description>
</annot>
```

Fig. 6. Example of a page-clipping annotation document

4 Automatic Generation of Annotation

In the page-clipping annotation language, annotation elements such as `<keep>`, `<remove>`, and `<insertattribute>` impose modifications of a target document, and can be regarded as the transformational annotations. On the other hand, in the other annotation languages, such as a page-splitting annotation language [11], the language provides assertional elements like `<role>` for specifying a role for an annotated element (as one of the values like proper content, advertisement, or decoration), and `<importance>` for specifying the priority of an annotated element in relation to the other elements in the same page.

Every annotation is an assertion in essence, but some annotations may further imply document transformation *with regard to the procedural semantics of a runtime engine*. Taking account of this distinction between annotations, it is possible to think about two approaches to annotation authoring: annotation by assertion, and annotation by transformation.

4.1 Annotation by Assertion

Existing annotation editors support the creation of assertional annotations by providing views easy to indicate a location to be annotated (e.g., an HTML browser view [1,5,8,20,23] and a WYSIWYG HTML editor [12,14]). We have

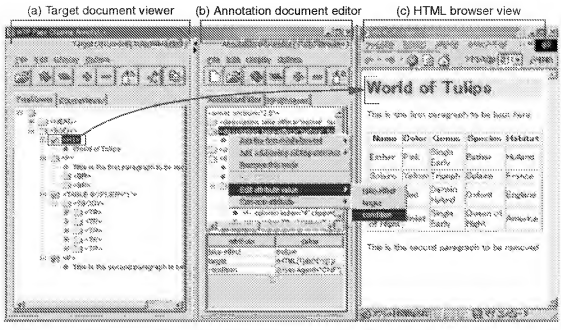

Fig. 7. Sample display of the authoring tool for assertional annotations

developed an annotation editor, which follows the annotation by assertion approach [1]. Fig. 7 shows a screen of the annotation editor.

This annotation editor can be customized for different annotation vocabularies as long as the language is compliant with a prescribed schema for a class of annotation languages [1]. With this annotation editor, a user can select a node to be annotated in either a target document viewer [Fig. 7(a)] or an HTML browser view [Fig. 7(c)] if the target document is an HTML page. The node selection in either view synchronizes with the other, and a user can create an annotation description by using a popup menu that appears when the user clicks the right mouse button. The details of the annotation content can be edited by using the menu-based annotation document editor [Fig. 7(b)], which can provide context-sensitive help based on the annotation language schema at hand. Since the page-clipping annotation language mentioned earlier is compliant with the prescribed schema, the annotation document shown in Fig. 6, for example, can be created with this annotation editor.

In this way, the annotation editor allows users to indicate a position to be annotated and declare properties as the content of an annotation. The HTML browser view, for example, is very helpful to indicate the position. However, the HTML page or target document remains unchanged, even when annotations specify transformation (e.g., insertion and removal) of the annotated portions.

4.2 Annotation by Transformation

By using a WYSIWYG editor, it is easy for the users to modify a target document toward the desired results of the customization. The annotation by transformation is an approach to generating the content customization metadata au-

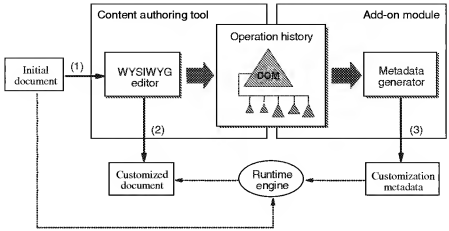

Fig. 8. Environment for generating transformational annotations

tomatically from the user's editing operations for the customization. An environment for generating transformational annotations is depicted in Fig. 8. With this metadata generation tool, first a user opens a document to be customized (e.g., an HTML file). The user then edits the document by using the full capabilities of the WYSIWYG authoring tool. The user's actions are recorded into an operation history while working in a recording mode. The user, however, does not have to care about the recording process behind the scenes. When the editing is finished, the user will have a customized document. At the same time, the metadata generator creates the customization metadata that can be used by a runtime engine to replicate the transformation from the initial document to the customized document.

Since users can perform a given editing task in different ways, it is necessary to prescribe a set of edit operations to be recorded in the user interface. The essential mechanism of the metadata generation depends solely on the Document Object Model (DOM) [7], and can be used for not only as arbitrary metadata format when a custom generator is provided, but also with an arbitrary content authoring tool as long as the editor adopts the DOM as an internal document model.

Fig. 9 shows a screenshot of the prototype environment that generates page-clipping annotations on the basis of users' editing operations with a WYSIWYG HTML editor [Fig. 9(a)]. The area of the WYSIWYG editor can be changed by selecting one of the sizes available from the page size button [Fig. 9(b)], for example, reducing the area in the screen shot 1/4 VGA size.

Fig. 9(c) shows the generated metadata, which is an XML document in the page-clipping annotation language. The look and feel of this WYSIWYG editor is exactly the same as in the original HTML editor, except for the small window for the generated annotations. A recording button [Fig. 9(d)] is used to start and stop recording a user's operations. While the toggle button remains depressed, the user's operations are recorded. When the button returns to its normal state, the recording stops. A page-clipping annotation document is created from the recorded operations by pressing the create annotation button [Fig. 9(e)].

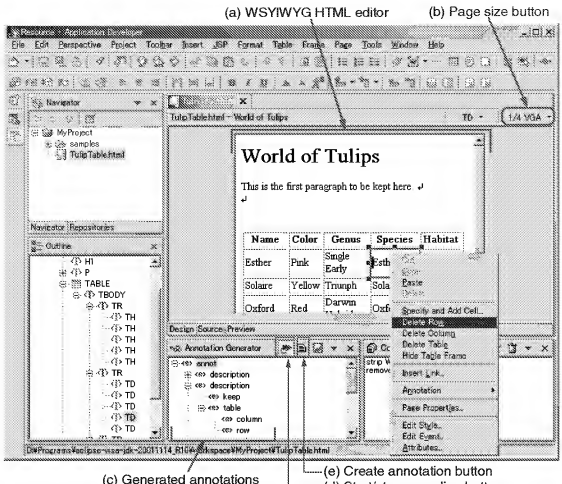

Fig. 9. Sample display from the environment for generating transformational annotations

Transformations for the page-clipping shown in Fig. 5 includes the removal of a table column and a row, and change of a table attribute value [Fig. 6(b)], as well as the removal of a paragraph [Fig. 6(c)]. All of the transformations can be done easily by using the HTML editor in a WYSIWYG manner. For example, a table row can be removed by selecting a cell in the row to be deleted and then choosing the menu item "Delete Row" from a popup menu (Fig. 9) that appears after clicking the right mouse button. The `cellpadding` attribute of the table can also be changed from a dialog window for the table attribute settings.

All the above-mentioned editing operations actually modify the target document by means of DOM manipulation operations, and can be created as transformational annotations. However, intrinsically assertional annotations cannot be created on the basis of the DOM manipulation operations, but need to be declared explicitly as assertions. Annotation editors that follow the annotation by assertion approach play a complementary role in such situations instead of the metadata generator for the transformational annotations.

```
<description take-effect="before" target="/HTML[1]/BODY[1]/H1[1]"
    condition="(User-Agent=*CNF*)">
  <remove/>
</description>

<description take-effect="after" target="/HTML[1]/BODY[1]/H1[1]"
    condition="(User-Agent=*CNF*)">
  <keep/>
</description>
```

Fig. 10. Examples of conditional annotations

An example of assertional annotation is given below in the page-clipping annotation language. Annotations may be applied conditionally depending on a delivery context, or depending on profiles of a client agent to be specified in the HTTP header field. By adding a `condition` attribute to a `<description>` element, the annotation will be executed only if the condition is true [26]. When it is necessary to remove the header element (`<H1>`) at the beginning of the sample document, the user needs to add a pair of annotations to start a remove state before the first H1 element and set the state back to 'keep' just after the H1 element. In addition, it is necessary for these annotations to be applied only if a client agent is given with a particular profile attribute value.

Fig. 10 shows examples of conditionally applied annotation descriptions. That is, the removal of the first H1 element is done only when a substring 'CNF' is included as a device type in the HTTP header. The asterisk ('*') here is a wild-card expression for the string matching. In addition, more complex conditions can be given by using conjunctive and/or disjunctive expression.

4.3 Metadata Generation Procedure

The DOM (Document Object Model) [7] defines a logical structure for documents, and provides ways of accessing and manipulating XML or well-formed HTML documents. The DOM represents a document as a hierarchy of node objects, which may be associated with attributes. DOM trees are changed by node insertion (*insertBefore, replaceChild, appendChild*), node removal (*removeChild*), or attribute change (*setAttribute, removeAttribute*). Since the DOM provides standardized, general-purpose operations for the transformation of DOM trees, it is reasonable to record users' operations as a sequence of DOM manipulation operations rather than as primitive operations such as a sequence of keystrokes and mouse clicks. Therefore, along with the notion of current node, which may be either a leaf node or an entire subtree, the model of WYSIWYG editing adopted here consists of the four basic operations: *insert* to add a subtree, *remove* to delete a subtree, *modify* to change a node, and *copy* to replicate a subtree. The rationale behind this editing model was already reported in another article [15].

Since the WYSIWYG editing for page clipping is typically done by removing the unnecessary portion of the original document, it is assumed that the clipping state of a created annotation document begins with 'keep' as shown in Fig. 6(a).

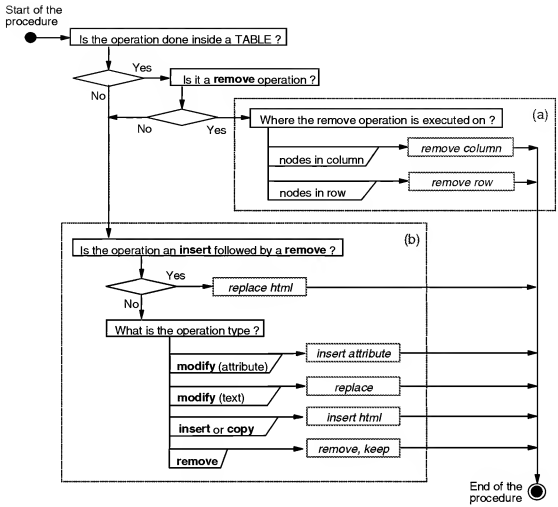

Fig. 11. Procedure for the generation of page-clipping annotations

Individual annotations are then created by examining the sequence of the user's recorded operations from the beginning to the end.

Fig. 11 depicts the procedure of annotation generation. In the figure, the terms in bold face are the primitive operations prescribed in the model of editing, while the terms in italics correspond to the elements of the page-clipping annotation language.

The generation procedure consists of the two sub-processes: one is to deal with manipulation with in a HTML table element, and the other is for other cases. When a focal node is a descendant of a TABLE element and its operation is removal [Fig. 11(a)], the generator creates annotations for removing the table rows or columns by using the `<column>` and `<row>` elements. In other cases [Fig. 11(b)], clipping annotations are created taking account of the distinction of the primitive operations (i.e., insert, remove, modify, and copy),

5 Concluding Remarks

Annotation by transformation is an innovative approach to helping users with the generation of content customization metadata, because it allows the users to create a customized result interactively with an example, rather than abstractly without any concrete example. Since users of the metadata generator do not have to learn the annotation language at all, this approach is particularly suitable for page designers or novice programmers who are not necessarily familiar with annotation languages. However, it is also useful for skilled designers too, because the environment allows users to concentrate on customizing Web pages without paying any attention to the metadata authoring task.

It has been pointed out that the success of an environment for programming by example depends far more on the user experience of interacting with the environment than the induction algorithms used to create the user's programs [4]. The advantage of annotation by transformation in this regard is that it does not rely on any particular editors, including a conventional WYSIWYG HTML editor, rather than a special editor tailored for the annotation generation. Further research will be needed to investigate the applicability of this technique. The main problem is to determine to what extent users are willing to accept the annotation by transformation environment that guesses what they are doing, and that occasionally might make inadequate or inappropriate generalizations. However, the approach proposed in this paper would be an important step towards the realization of a full-fledged support tool for a device-agnostic authoring environment.

References

1. Abe, M. and Hori, M.: A visual approach to authoring XPath expressions. *Proceedings of Extreme Markup Languages 2001*, pp. 1–14 Montréal, Canada (2001).
2. Bickmore, T. W. and Schilit, B. N.: Digestor: Device-independent access to the World Wide Web. *Proceedings of the 6th International World Wide Web Conference*, Santa Clara, CA (1997).
3. Butler, M. H., Current Technologies for Device Independence. Technical Report HPL-2001-83, Hewlett-Packard Company (2001).
4. Sypher, A.: Introduction: Bringing programming to end users. In Allen Sypher *et al.* (Eds.): *Watch What I Do: Programming by Demonstration.* pp. 1–11, The MIT Press, Cambridge, MA (1993).
5. Denoue, L. and Vignollet, L.: An annotation tool for Web browsers and its applications to information retrieval. *Proceedings of the 6th Conference on Content-Based Multimedia Information Access (RIAO 2000)*, Paris, France (2000).
6. Device Independence Principles. *W3C Working Draft*, http://www.w3.org/TR/di-princ/ (2001).
7. Document Object Model (DOM) Level 1 Specification Version 1.0. *W3C Recommendation*, http://www.w3.org/TR/REC-DOM-Level-1/ (1998).
8. Erdmann, M., Maedche, A., Schnurr, H.-P., and Staab, S.: From manual to semi-automatic semantic annotation: about ontology-based text annotation tools. *Proceedings of the COLING 2000 Workshop on Semantic Annotation and Intelligent Content*, Luxembourg (2000).

9. Fox, A. and Brewer, E. A.: Reducing WWW latency and bandwidth requirements by real-time distillation. *Proceedings of the 5th International World Wide Web Conference*, Paris, France (1996).
10. Heflin, J. and Hendler, J.: Semantic interoperability on the Web. *Proceedings of Extreme Markup Languages 2000*, pp. 111–120 (2000).
11. Hori, M., Kondo, G., Ono, K., Hirose, S., and Singhal, S.: Annotation-based Web content transcoding. *Proceedings of the 9th International World Wide Web Conference (WWW9)*, pp. 197–211, Amsterdam, Netherlands (2000).
12. Hori, M., Ono, K., Kondo, G., and Singhal, S.: Authoring tool for Web content transcoding. *Markup Languages: Theory & Practice*, **2**(1): 81–106 (2000).
13. Hori, M.: Semantic annotation for Web content adaptation. In D. Fensel, J. Hendler, H. Lieberman, and W. Whalster (Eds), *Spinning the Semantic Web*, pp. 542–573, MIT Press, Boston, MA (2002).
14. Kahan, J. and Koivunen, M.-R.: Annotea: an open RDF infrastructure for shared Web annotations. *Proceedings of the 10th International World Wide Web Conference (WWW10)*, pp. 623–632, Hong Kong (2001).
15. Koyanagi, T., Ono, K., and Hori, M.: Demonstrational Interface for XSLT Stylesheet Generation. *Markup Languages: Theory & Practice*, **2**(2): 133–152 (2001).
16. Lassila, O.: Web metadata: a matter of semantics. *IEEE Internet Computing*, **2**(4): 30–37 (1998).
17. Lieberman, H. (Ed.): *Your Wish is My Command: Programming by example*. Morgan Kaufmann Publishers, San Francisco (2001).
18. Maes, S. H. and Raman, T. V.: Position paper for the W3C/WAPWorkshop on the Multi-modal Web, W3C, http://www.w3.org/2000/09/Papers/IBM.html (2000).
19. Mea, V. D., Beltrami, C. A., Roberto, V., and Brunato, D.: HTML generation and semantic markup for telepathology. *Proceedings of the 5th International World Wide Web Conference (WWW5)*, pp. 1085–1094, Paris, France (1996).
20. Nagao, K., Shirai, Y., and Kevin, S.: Semantic annotation and transcoding: making Web content more accessible. *IEEE Multimedia*, **8**(2): 69–81 (2001).
21. Ono, K., Koyanagi, T., Abe, M. and Hori, M.: XSLT Stylesheet Generation by Example with WYSIWYG Editing. *Proceedings of the International Symposium on Applications and the Internet (SAINT 2002)*, pp. 150–159 (2002).
22. Rousseau, J. F., Macias, A. G., de Lima, J. V., and Duda, A.: User adaptable multimedia presentations for the World Wide Web. *Proceedings of the 8th International World Wide Web Conference*, pp. 195–212, Toronto, Canada (1999).
23. Sakairi, T. and Takagi, H.: An annotation editor for nonvisual Web access. *Proceedings of the 9th International Conference on Human-Computer Interaction (HCI International 2001)*, pp. 982–985, New Orleans, LA (2001).
24. Spinks, R., Topol, B., Seekamp, C., and Ims, S.: Document clipping with annotation. developerWorks, IBM Corp.
http://www.ibm.com/developerworks/ibm/library/ibm-clip/ (2001).
25. Eclipse Platform. *eclipse.org Consortium*, http://www.eclipse.org/ (2002).
26. WebSphere Transcoding Publisher Version 4.0 Developer's Guide. IBM Corp. (2001).
27. XML Path Language (XPath) Version 1.0.
W3C Recommendation, http://www.w3.org/TR/xpath (1999).
28. XSL Transformations (XSLT) Version 1.0.
W3C Recommendation, http://www.w3.org/TR/xslt (1999).